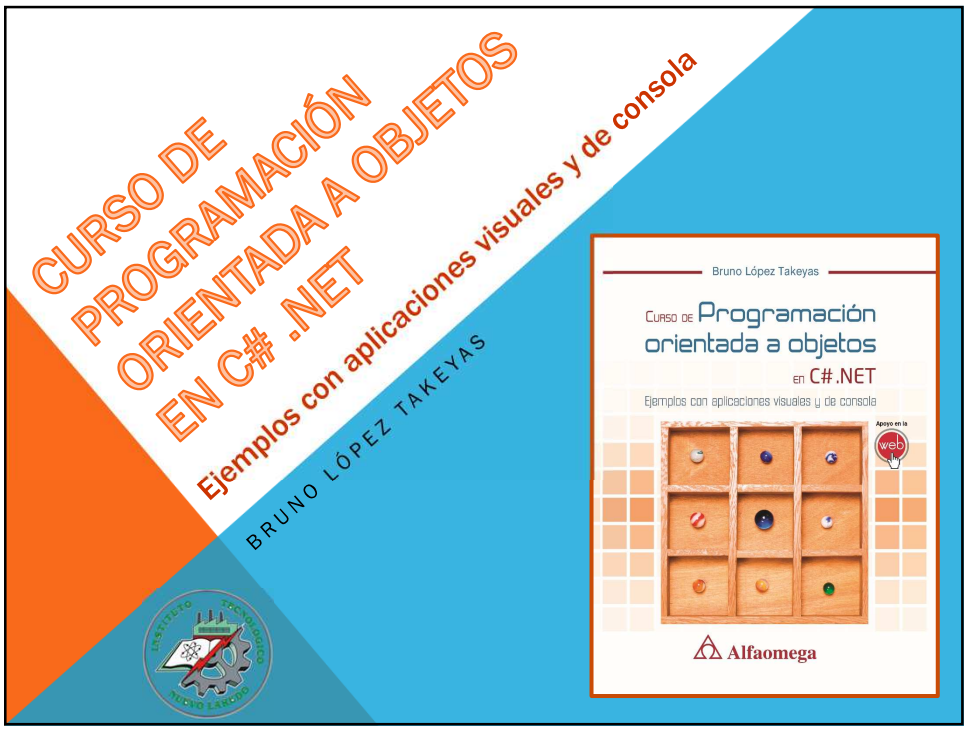
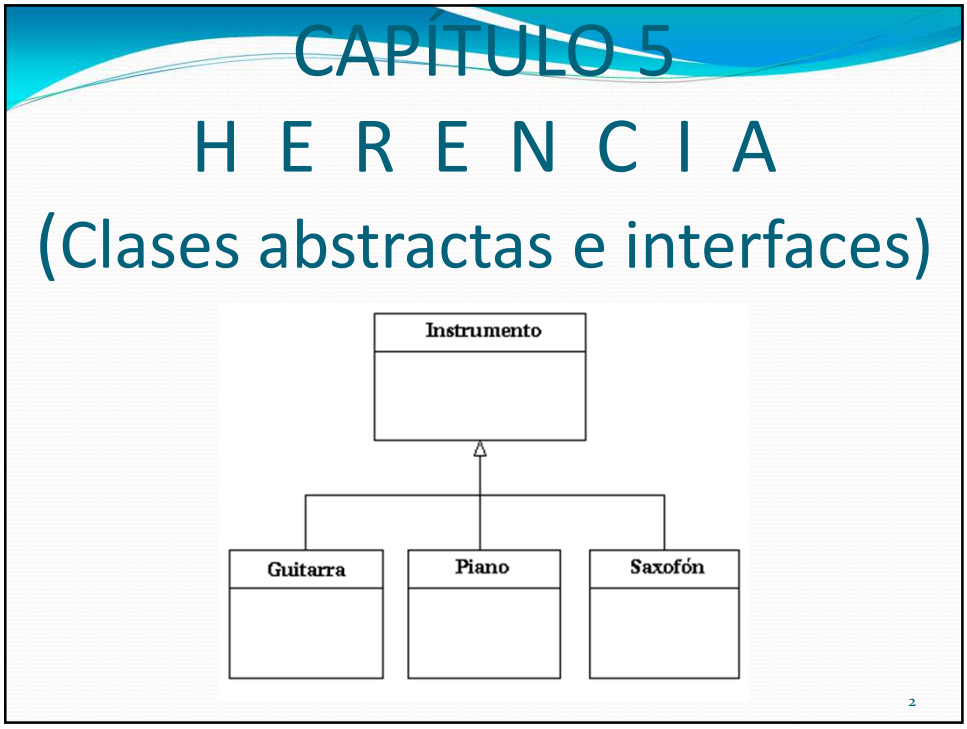

CURSO DE PROGRAMACIÓN ORIENTADA A OBJETOS EN C# .NET
Ejemplos con aplicaciones visuales y de consola
BRUNO LÓPEZ TAKEYAS
Bruno López Takeyas
Curso de Programación orientada a objetos en C#.NET
Ejemplos con aplicaciones visuales y de consola
Apoyo en la web
Alfaomega


CAPÍTULO 5
H E R E N C I A
(Clases abstractas e interfaces)
Instrumento
Guitarra
Piano
Saxofón
2

Preguntas detonadoras

- Parece paradójico que una clase no pueda crear objetos a partir de ella, ¿realmente lo es?
- Si una clase abstracta no puede generar instancias, ¿entonces para qué sirve?
- Si un miembro abstracto no tiene implementación, ¿entonces para qué sirve?
- En una clase abstracta, ¿todos sus miembros son abstractos?
- ¿En qué se parece una interfase a una clase abstracta? ¿En qué difieren?
- ¿Se puede definir un miembro abstracto dentro de una clase no abstracta?

3

Clases abstractas e interfaces

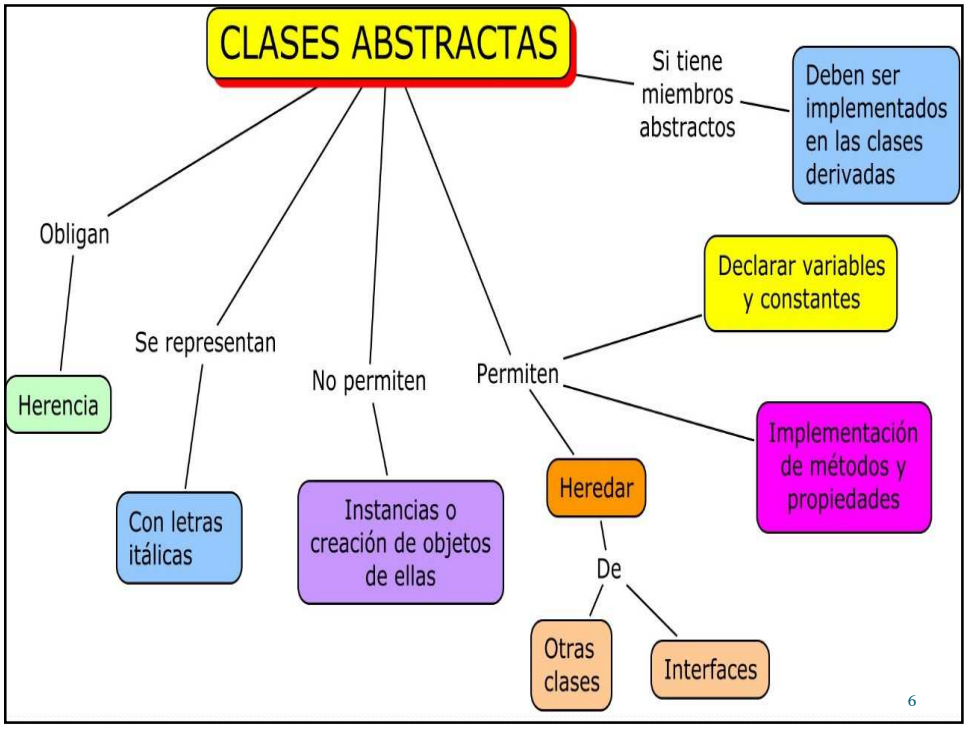
- Tanto las **clases abstractas** como las **interfaces** son mecanismos que obligan la herencia
- No se pueden instanciar, es decir, no se puede crear objetos de ellas
- Si no se pueden crear objetos a partir de ellas, ¿entonces para qué sirven?

4

Clases abstractas

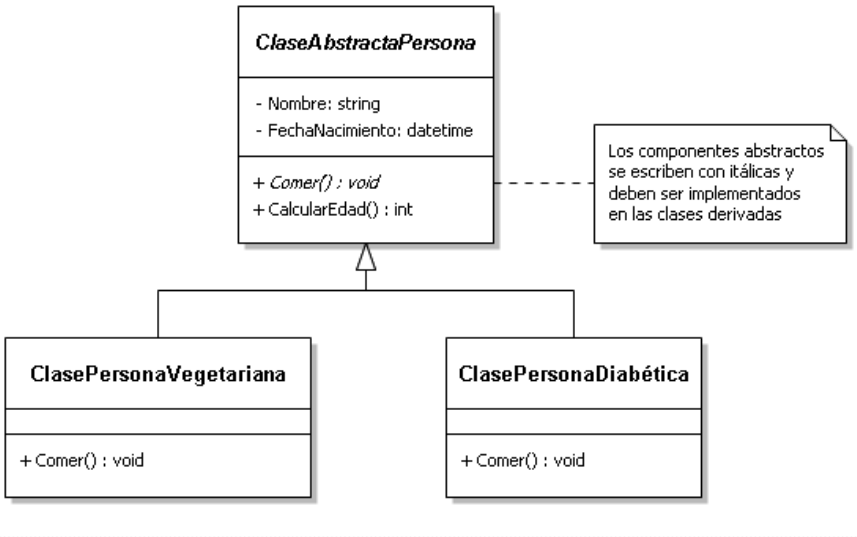
- Son mecanismos que obligan la herencia
- No se pueden instanciar, es decir, no se puede crear objetos de ellas
- Se utilizan solamente para heredar de ellas (Forzar u obligar la herencia).
- Se antepone la palabra "**`abstract`**" al nombre de la clase.

5



6

Ejemplo de clase abstracta



Implementación de una clase abstracta en C#

```
public abstract class ClaseAbstractaPersona
{
   string Nombre;
   DateTime FechaNacimiento;

   public abstract void Comer();

   public int CalcularEdad()
   {
      //Aquí se implementa este método
   }
}
```

Método abstracto (debe ser implementado en las clases derivadas)

8

Ejemplo: Clase Abstracta

```
abstract class Persona
{
    private string _strNombre;
    private string _strApellido;
    public string Nombre
    {
        get { return _strNombre; }
        set { _strNombre = value; }
    }
    public string Apellido
    {
        get { return _strApellido; }
        set { _strApellido = value; }
    }
    public string ObtenerNombreCompleto()
    {
      return
         this.Nombre +" "+ this.Apellido;
    }
}
```

```
class Empleado : Persona
{
    private int _intClaveEmpleado;
    public int Clave
    {
     get { return _intClaveEmpleado; }
     set { _intClaveEmpleado = value; }
    }
}
class Cliente : Persona
{
    private string _strRfc;
    public string RFC
    {
        get { return _strRfc; }
        set { _strRfc = value; }
    }
}
```

9

Continuación… Ejemplo de Clase Abstracta

```
class Programa
{
    static void Main()
    {
        Empleado unEmpleado = new Empleado();
        unEmpleado.Nombre = "Juan";
        unEmpleado.Apellido = "Gonzalez";
        unEmpleado.Clave = 1;
        System.Console.WriteLine(unEmpleado.ObtenerNombreCompleto());

        Cliente unCliente = new Cliente();
        unCliente.Nombre = "Pedro";
        unCliente.Apellido = "Ramirez";
        unCliente.RFC = "RAHP780212";
        System.Console.WriteLine(unCliente.ObtenerNombreCompleto());

        System.Console.ReadLine();
    }
}
```

10

Clases abstractas con elementos abstractos

- Las clases abstractas ***pueden*** definir métodos y propiedades abstractos, con lo que su respectiva implementación en la subclase es obligatoria. (Los elementos abstractos ***deben*** ser sobrescritos en la subclase).
- Se utiliza "`abstract`" para definir elementos abstractos (solo dentro de clases abstractas).
- Los elementos abstractos **NO** proporcionan implementación; solo declaraciones.
- En la subclase, se utiliza "`override`" para realizar la implementación correspondiente.

11

Miembros abstractos

- Una clase abstracta ***puede*** tener datos (atributos) e implementar métodos y propiedades como una clase normal y además ***puede*** tener miembros abstractos (métodos o propiedades).
- Los miembros abstractos **NO** tienen implementación (están vacíos).
- ¿Para qué sirve un método vacío o propiedad vacía y que no realiza acciones?

12

Ejemplo: Clase abstracta con elementos abstractos

```
abstract class Persona
{
    private string _strNombre;
    private string _strApellido;
    public string Nombre
    {
        get { return _strNombre; }
        set { _strNombre = value; }
    }
    public string Apellido
    {
        get { return _strApellido; }
        set { _strApellido = value; }
    }
    public abstract int Clave
    { get; set; }

    public abstract string ConsultarTodosLosDatos();

    public string ObtenerNombreCompleto()
    {
        return
        this.Nombre + " " + this.Apellido;
    }
}
```

Se **DEBEN** implementar estos elementos al heredar de esta clase.

13

Clase abstracta con elementos abstractos (cont.)

```
class Empleado : Persona
{
    private int _intClaveEmpleado;
    public override int Clave
    {
        get { return _intClaveEmpleado; }
        set { _intClaveEmpleado = value; }
    }
    public override string ConsultarTodosLosDatos()
    {
        return "------Datos del Empleado: \n" +  this.Clave + " " +
               this.Nombre + " " + this.Apellido;
    }
}

class Cliente : Persona
{
    private int _intClaveCliente;
    public override int Clave
    {
        get { return _intClaveCliente; }
        set { _intClaveCliente = value; }
    }
    public override string ConsultarTodosLosDatos()
    {
        return "******Datos del Cliente: \n" + this.Clave + " " +
                this.Nombre + " " + this.Apellido;
    }
}
```

Implementación

Implementación

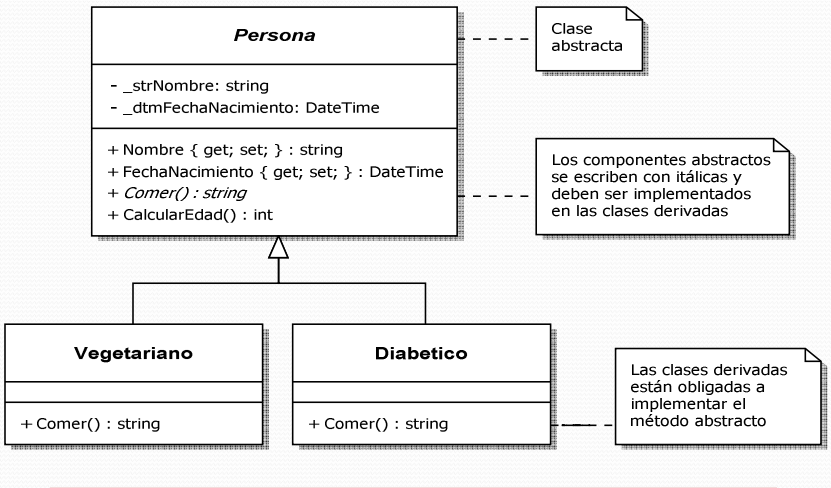
Clase abstracta con elementos abstractos (cont.)

```
class Programa
{
    static void Main()
    {
        Empleado unEmpleado = new Empleado();
        unEmpleado.Nombre = "Juan";
        unEmpleado.Apellido = "Gonzalez";
        unEmpleado.Clave = 1;
        System.Console.WriteLine( unEmpleado.ConsultarTodosLosDatos() );
        System.Console.WriteLine( unEmpleado.ObtenerNombreCompleto() );

        Cliente unCliente = new Cliente();
        unCliente.Nombre = "Pedro";
        unCliente.Apellido = "Ramirez";
        unCliente.Clave = 34;
        System.Console.WriteLine( unCliente.ConsultarTodosLosDatos() );
        System.Console.WriteLine( unCliente.ObtenerNombreCompleto());
        System.Console.ReadLine();
    }
}
```

15

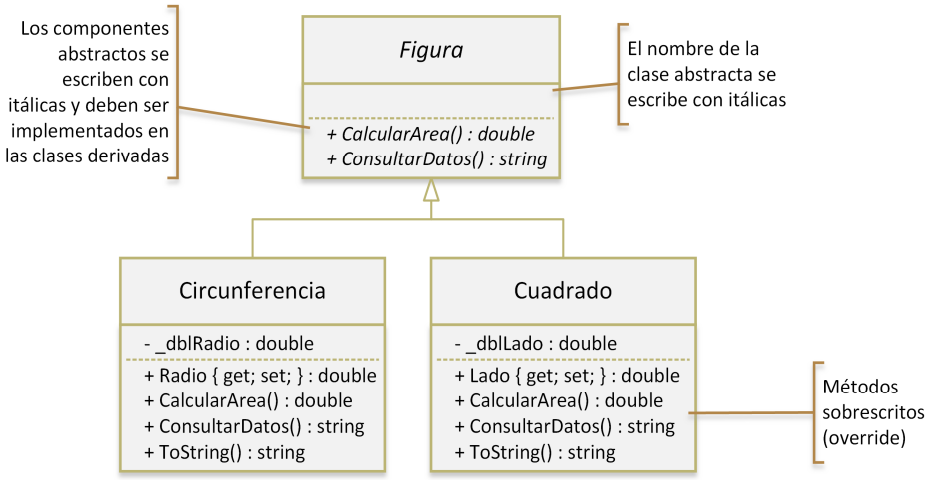
Miembros abstractos



En UML las clases y sus miembros abstractos se escriben con itálicas y en C# .NET se codifican anteponiendo la palabra “abstract”

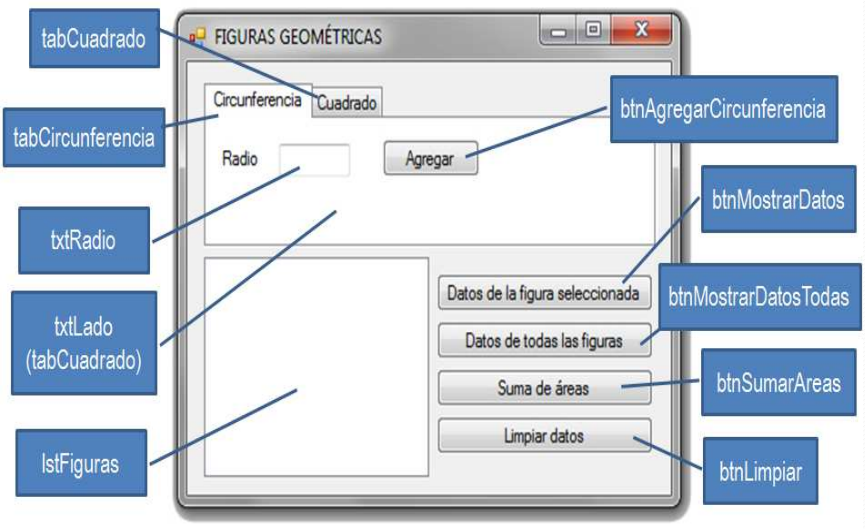
16

Prog. 5.5.- Clase abstracta con métodos abstractos



17

Diseño de la forma

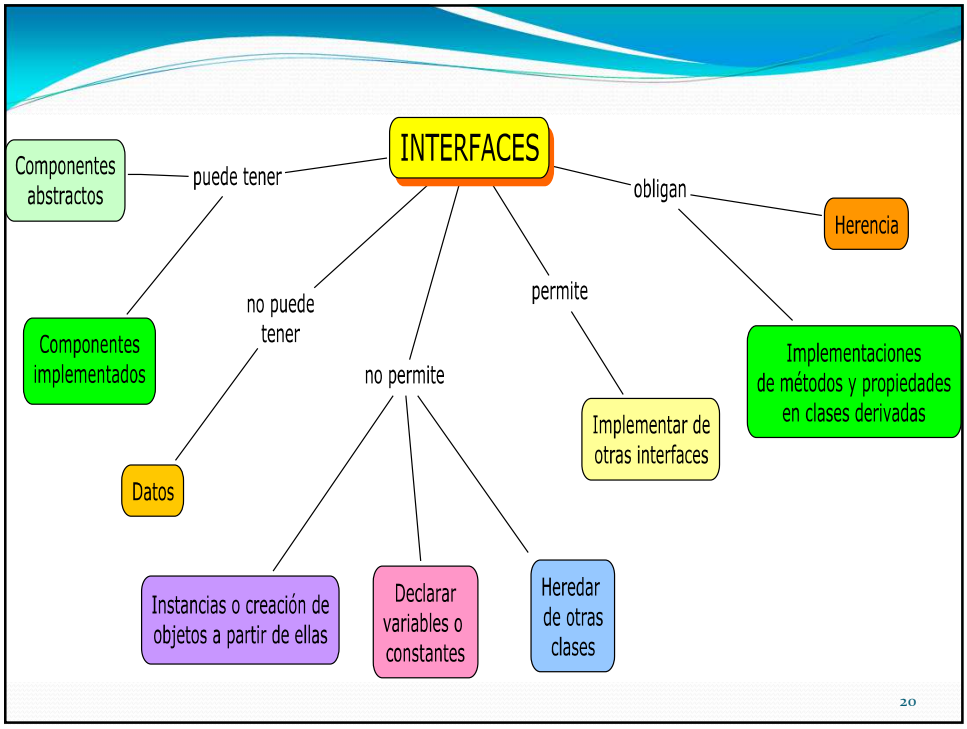


18

Interfaces

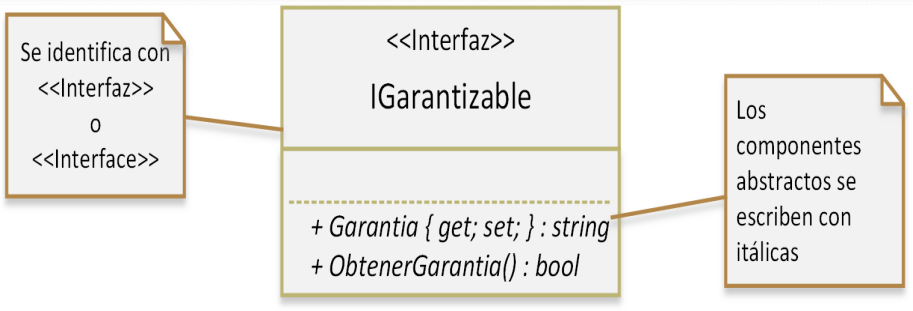
- Son mecanismos para que puedan interactuar varios objetos no relacionados entre sí
- Son protocolos o "contratos" que obligan la herencia
- Las interfaces **pueden** tener:
  - *Componentes abstractos*
  - *Componentes implementados (a partir de la versión 8.0 de C#)*
- Al igual que las clases abstractas, son plantillas de comportamiento que deben ser implementados por otras clases.

19



20

Ejemplo de una interfase



En UML una interfase se identifica con <<Interfaz>> o <<Interface>>

21

Notas acerca de las interfaces

- Una clase que herede de una interfase **debe** implementar **todas** las definiciones contenidas en ella.
- Los elementos de la interfase no llevan los modificadores "`public`" o "`abstract`".

NOTA: A partir de C# 8.0 ya se permite que las interfaces tengan métodos y/o propiedades con implementación (no estén vacíos)

22

Versiones

Requisitos previos

Deberá configurar la máquina para ejecutar .NET Core, incluido el compilador de C# 8.0. El compilador de C# 8.0 está disponible a partir de la versión 16.3 de Visual Studio 2019 o del SDK de .NET Core 3.0.

Target framework	version	C# language version default
.NET	6.x	C# 10
.NET	5.x	C# 9.0
.NET Core	3.x	C# 8.0
.NET Core	2.x	C# 7.3
.NET Standard	2.1	C# 8.0
.NET Standard	2.0	C# 7.3
.NET Standard	1.x	C# 7.3
.NET Framework	all	C# 7.3

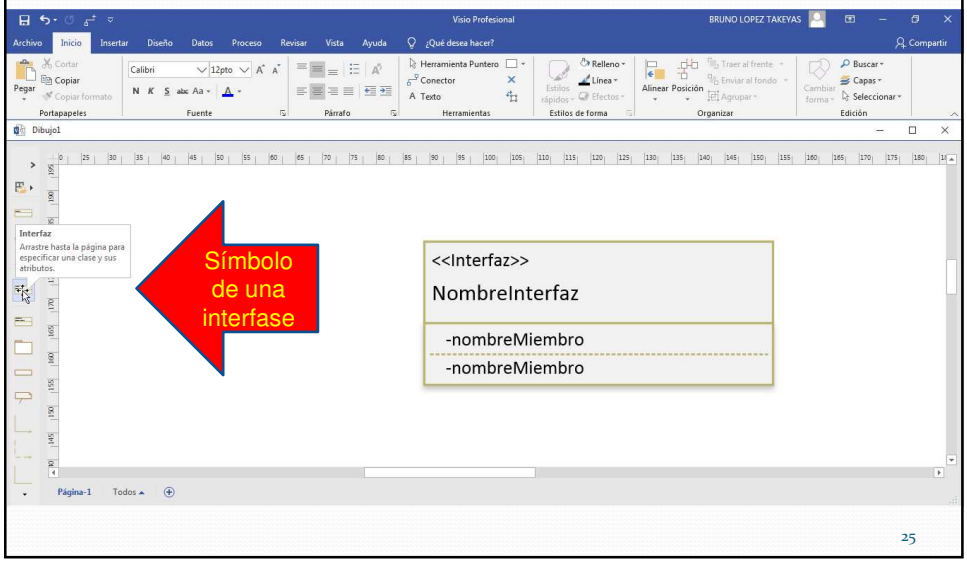
23

Nomenclatura de las interfaces

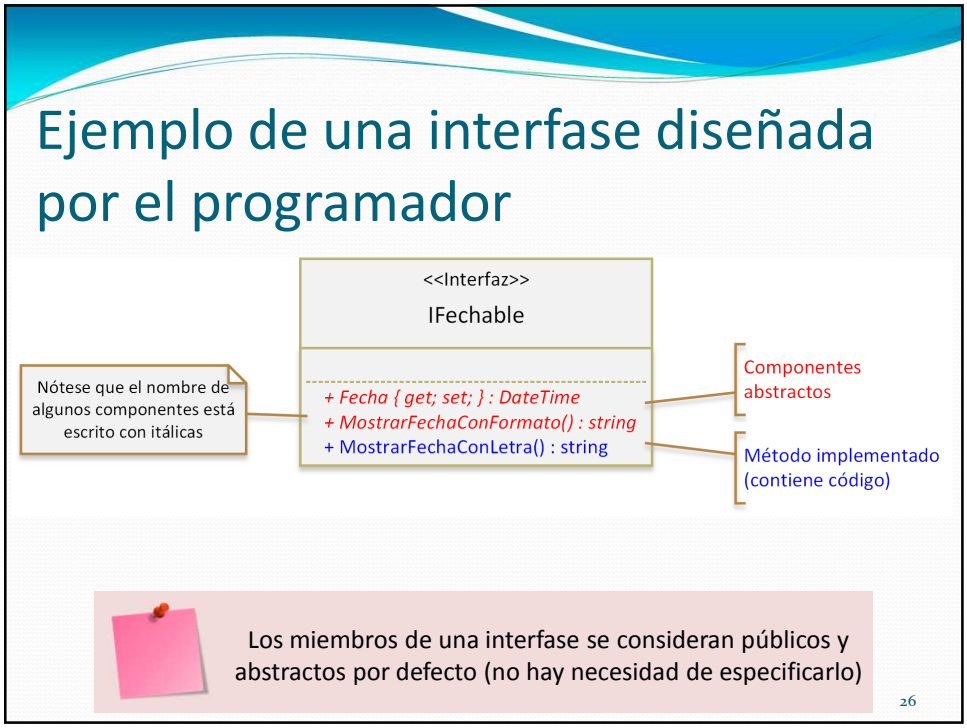
- El nombre de la interfase debe iniciar con la letra "I"
- Se recomienda que termine con "...able"
- Ejemplos:
  - *IGarantizable*
  - *IFechable*
  - *IComprobable*
  - *IMostrable*

24

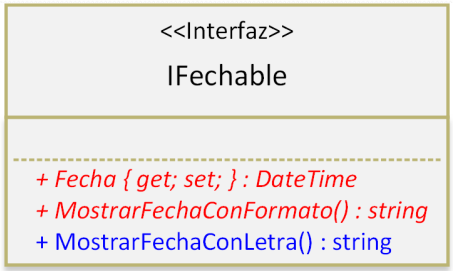
Interfaces en MS Visio



Ejemplo de una interfase diseñada por el programador



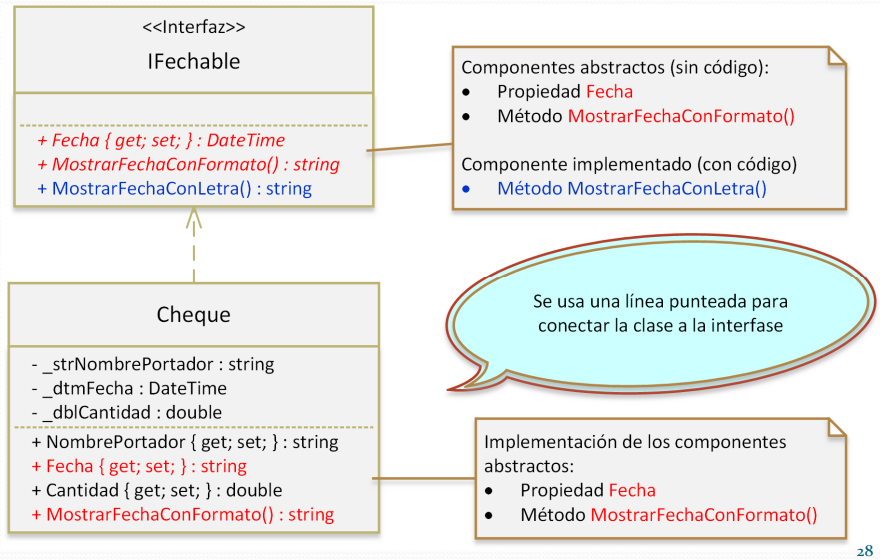
Ejemplo de una interfase diseñada por el programador



- Obtiene la fecha del sistema mediante `DateTime.Now`
- Obliga a implementar los métodos abstractos que contiene en las clases derivadas de ella.
- Se deben implementar **todos** sus componentes abstractos, de lo contrario nos indica un error.

27

Uso de una interfase diseñada por el programador



28

Codificación de la interfase

```
interface IFechable
    {
        DateTime Fecha { get; set; }  // Propiedad abstracta
        string MostrarFechaConFormato(); // Método abstracto

        public string MostrarFechaConLetra()
        {
            string strFechaConLetra = Fecha.Day.ToString() + " de ";
            switch (Fecha.Month)
            {
                case 1: strFechaConLetra += "enero"; break;
                case 2: strFechaConLetra += "febrero"; break;
                case 3: strFechaConLetra += "marzo"; break;
                case 4: strFechaConLetra += "abril"; break;
                case 5: strFechaConLetra += "mayo"; break;
                case 6: strFechaConLetra += "junio"; break;
                case 7: strFechaConLetra += "julio"; break;
                case 8: strFechaConLetra += "agosto"; break;
                case 9: strFechaConLetra += "septiembre"; break;
                case 10: strFechaConLetra += "octubre"; break;
                case 11: strFechaConLetra += "noviembre"; break;
                case 12: strFechaConLetra += "diciembre"; break;
            }
            strFechaConLetra += " de " + Fecha.Year;
            return (strFechaConLetra);
        }
    }
```

Componentes abstractos

Método implementado

Uso de una interfase en C#

```
class Cheque: IFechable
{
    private string _strNombrePortador;
    public string NombrePortador
    {
       get { return _strNombrePortador; }
       set { _strNombrePortador = value; }
    }

    private DateTime _dtmFecha;
    public DateTime Fecha  // Implementación de la propiedad de IFechable
    {
       get { return _dtmFecha; }
       set { _dtmFecha = value; }
    }

    private double _dblCantidad;
    public double Cantidad
    {
       get { return _dblCantidad; }
       set { _dblCantidad = value; }
    }

    public string MostrarFechaConFormato() // Implementación del método de IFechable
    {
       return Fecha.ToShortDateString();
    }
}
```

La clase Cheque implementa la interfase IFechable

Implementación de los componentes de la interfase IFechable

Uso de un componente implementado en una interfase

```
((IFechable) unCheque).MostrarFechaConLetra();
```

Nombre de la Interfase

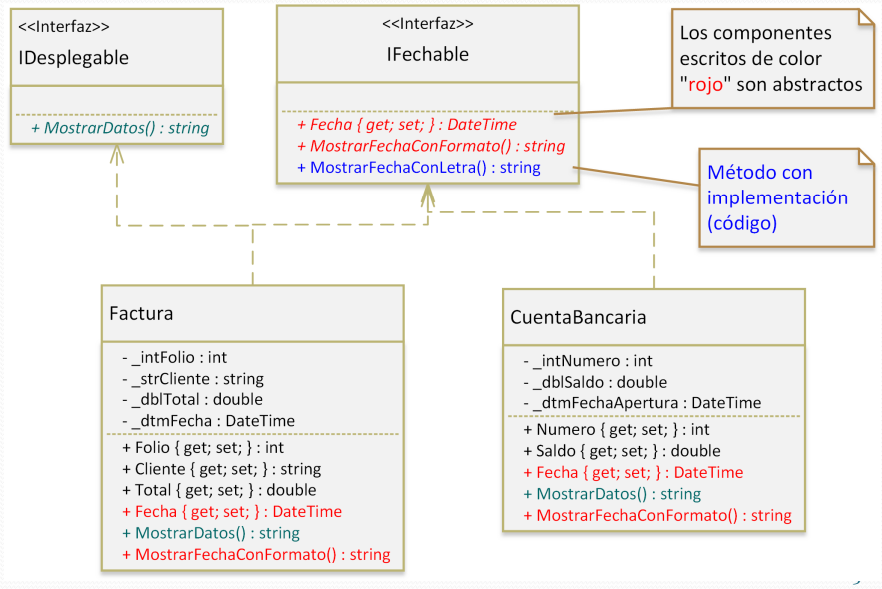
Nombre del objeto

Nombre del método implementado en la interfase

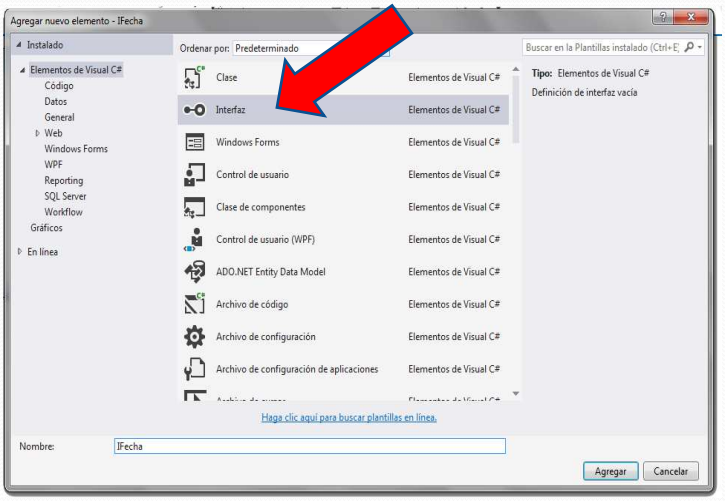
El método MostrarFechaConLetra() ya estaba implementado en la interfase IFechable (ya tenía el código)

31

Otro ejemplo de uso de interfaces



¿Cómo agregar una interfase al proyecto?



33

Ejemplo de una interfase

```
interface IVehiculo                    //Declaraciones solamente
{
    string Marca
    {
        get;
        set;
    }
    void Arrancar();
}

class Carro: IVehiculo                 //Implementación de toda la interfase
{
    private string _strMmarca;
    public string Marca
    {
        get { return _strMarca; }
        set { _strMarca = value;  }
    }
    public void Arrancar()
    {
        System.Console.WriteLine("Arrancar....Clase Carro");
    }
}
```

34

Ejemplo: Heredando de una clase e implementando dos interfaces

```
interface ICuadrado
{
   double Lado
   {
      get;
      set;
   }
}

interface IFiguraOperaciones
{
   double CalcularArea();
   double CalcularPerimetro();
}

public class Figura
{
   public virtual string ConsultarDatos()
   {
      return "Datos de la Figura: ";
   }
}
```

```
class Cuadrado :  Figura, ICuadrado,
                  IFiguraOperaciones
   {
       private double _dblLado;
       public double Lado
       {
           get { return _dblLado;  }
           set { _dblLado = value; }
       }
       public double CalcularArea()
       {
           return Lado * Lado;
       }
       public double CalcularPerimetro()
       {
           return 4 * Lado;
       }
       public override string ConsultarDatos()
       {
           return " Datos : Lado = " + Lado;
       }
   }
```

35

Ejemplo: Heredando de una clase e implementando dos interfaces (cont.)

```
class Program
   {
       static void Main()
       {
           Cuadrado c = new Cuadrado();
           c.Lado = 2;
           System.Console.WriteLine( c.ConsultarDatos() );
           System.Console.WriteLine("Area: " + c.CalcularArea());
           System.Console.WriteLine("Perimetro: " + c.CalcularPerimetro());
           System.Console.ReadLine();
       }
   }
```

36

Interfaces en C#

- IComparable
- IEquatable
- IEnumerator
- IEnumerable
- INotifyPropertyChanged
- Y otras …

37

La interfase IComparable

- Contiene la declaración del método CompareTo()

```
interface IComparable
{
   int CompareTo(object obj);
}
```

- El método CompareTo() devuelve un valor entero como resultado de la comparación

38

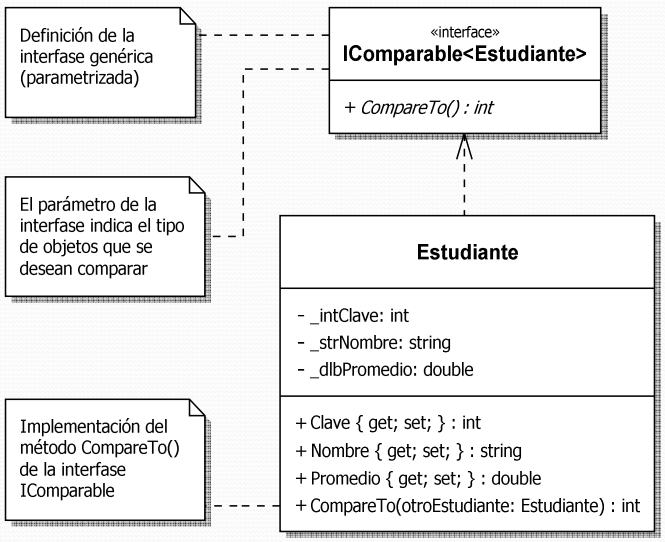
La función CompareTo()

```
int CompareTo(obj)
```

-1	Si this < obj
0	Si this == obj
1	Si this > obj

39

Uso de IComparable



40

Implementación de IComparable

```
class Estudiante : IComparable<Estudiante>

{
    private int _intClave;            // Atributos privados
    private string _strNombre;
    private double _dlbPromedio;

    public int Clave  {               // Propiedades públicas
           get { return _intClave; }
           set { _intClave = value; }
    }

    public string Nombre {
           get { return _strNombre; }
           set { _strNombre = value; }
    }

    public double Promedio {
           get { return _dlbPromedio; }
           set { _dlbPromedio = value; }
    }

    // Implementación del método CompareTo de la interfase IComparable
    public int CompareTo(Estudiante otroEstudiante)  {
           // Se utiliza el promedio de los estudiantes para determinar el orden
       if (this.Promedio > otroEstudiante.Promedio)
           return (1);
       else
           if (this.Promedio < otroEstudiante.Promedio)
             return (-1);
           else
             return (0);
     }
 }
```

¿Cómo comparar datos de tipo *string*?

```
class Estudiante : IComparable<Estudiante>

   {
       private int _intClave;             // Atributos privados
       private string _strNombre;
       private double _dlbPromedio;

       public int Clave {                 // Propiedades públicas
           get { return _intClave; }
           set { _intClave = value; }
       }

       public string Nombre {
           get { return _strNombre; }
           set { _strNombre = value; }
       }

       public double Promedio {
           get { return _dlbPromedio; }
           set { _dlbPromedio = value; }
       }

       // Implementación del método CompareTo de la interfase IComparable
       public int CompareTo(Estudiante otroEstudiante)  {
           return(this.Nombre.CompareTo(otroEstudiante.Nombre));
       }
   }
```

42

¿Un CompareTo() dentro de otro?

```
class Estudiante : IComparable<Estudiante>
   {

       . . .
       . . .
       . . .

       // Implementación del método CompareTo de la interfase Icomparable

       public int CompareTo(Estudiante otroEstudiante)
       {
           return(this.Nombre.CompareTo(otroEstudiante.Nombre));
       }
   }
```

- El `CompareTo()` de la clase `Estudiante` invoca al `CompareTo()` de la clase *`String`* (puesto que el `Nombre` es un dato de tipo cadena).

43

La interfase IEquatable

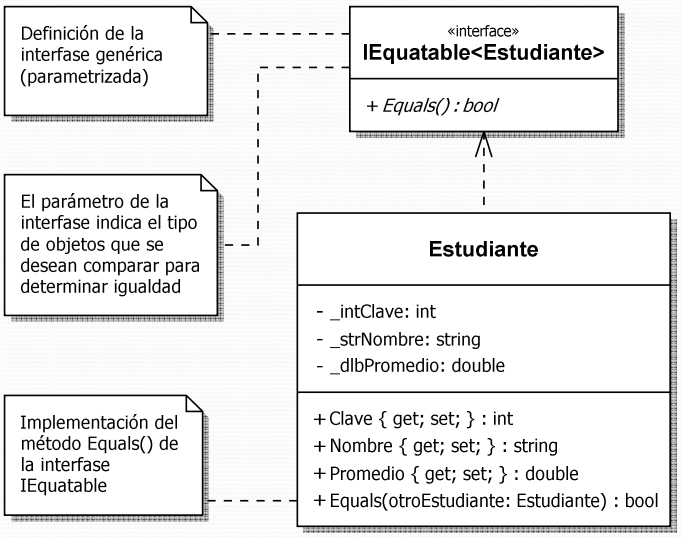
- Contiene la declaración del método `Equals()`

```
interface IEquatable<T>
{
        bool Equals(T obj);
}
```

- El método `Equals()` devuelve un valor booleano como resultado de la comparación

44

Uso de IEquatable



45

Implementación de IEquatable

```
class Estudiante : IEquatable<Estudiante>

{
     private int _intClave;              // Atributos privados
     private string _strNombre;
     private double _dlbPromedio;

     public int Clave {                  // Propiedades públicas
           get { return _intClave; }
           set { _intClave = value; }
     }

     public string Nombre  {
           get { return _strNombre; }
           set { _strNombre = value; }
     }

     public double Promedio  {
           get { return _dlbPromedio; }
           set { _dlbPromedio = value; }
     }

     // Implementación del método Equals de la interfase IEquatable
     public bool Equals(Estudiante otroEstudiante)  {
        // Se utiliza la clave de los estudiantes para determinar
        // si dos objetos son iguales
           return (this.Clave == otroEstudiante.Clave);
     }
}
```

46

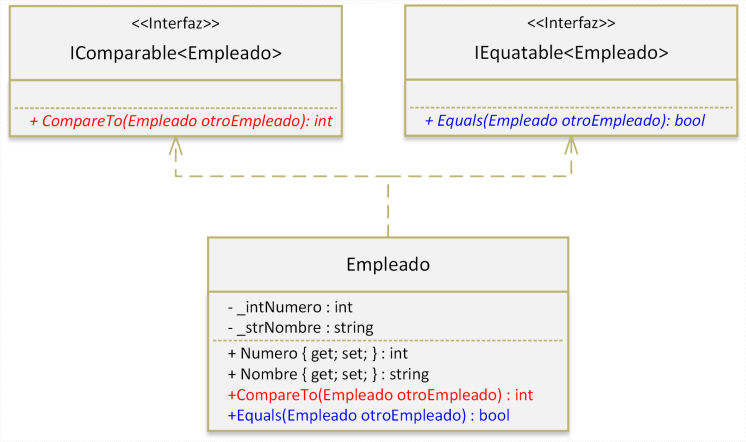
NOTA IMPORTANTE

- Las interfaces `IEquatable` e `IComparable` solamente comparan objetos del **mismo tipo**.
- No se pueden comparar objetos de diferentes tipos; es decir, creados a partir de clases diferentes.

Puesto que los objetos pueden realizar acciones, entonces tienen la capacidad de compararse entre sí para determinar si son iguales o para definir un orden específico a través de las interfaces IEquatable e IComparable

47

Uso de las interfaces IComparable e IEquatable



Al invocar los métodos `Equals()` y `CompareTo()` se hace una comparación a nivel objeto y no es necesario especificar cuál es el dato que se utiliza para hacer la comparación. Es la implementación de dichos métodos en la clase la que determina el criterio de comparación.

Implementación

```
class Empleado: IComparable<Empleado>, IEquatable<Empleado>
  {
        . . . .  // Atributos y propiedades

        public int CompareTo(ClaseEmpleado otroEmpleado)
        {
            return(this.Nombre.CompareTo(otroEmpleado.Nombre);
        }

        public bool Equals(ClaseEmpleado otroEmpleado)
        {
            return (this.Numero == otroEmpleado.Numero);
        }
   }
```

Use el método CompareTo() para comparar datos de tipo string

Ejemplo de uso

- Declaración e inicialización de los objetos:

```
Empleado miSecretaria = new Empleado();
Empleado miIntendente = new Empleado();

miSecretaria.Numero = 2;
miSecretaria.Nombre = "Rosa";

miIntendente.Numero = 3;
miIntendente.Nombre = "Luis";
```

50

Ejemplo de uso (cont.)

```
int intResultado = miSecretaria.CompareTo(miIntendente);

switch (intResultado) {
  case -1: MessageBox.Show("Nombre de la Secretaria < nombre del Intendente");
                  break;
  case 0: MessageBox.Show("Nombre de la Secretaria == nombre del Intendente");
                  break;
  case 1: MessageBox.Show("Nombre de la Secretaria > nombre del Intendente");
                  break;
}

if (miSecretaria.Equals(miIntendente))
    MessageBox.Show("Número de la Secretaria == número del Intendente");
else
    MessageBox.Show("Número de la Secretaria != número del Intendente");
```

51

CUESTIONARIO SYNESI2

Contestar:

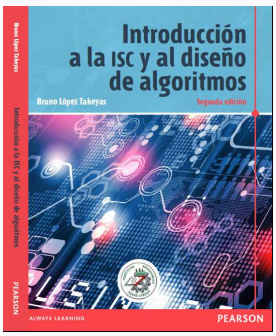
Cuestionario 4.1.-
Polimorfismo

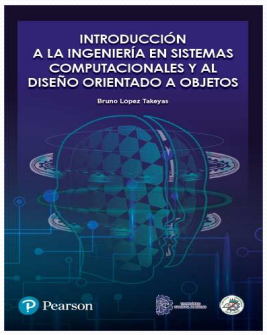
SYNESI
Instituto Tecnológico de Nuevo Laredo

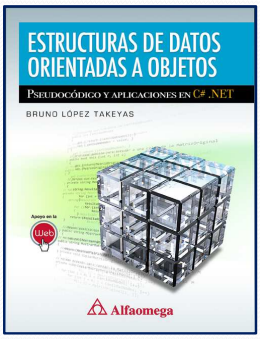
52

Otros títulos del autor

http://www.itnuevolaredo.edu.mx/Takeyas/Libro







bruno.lt@nlaredo.tecnm.mx

Bruno López Takeyas